8: Testing While Refactoring to Multiple Platforms

When we started developing this cookbook I told you that we were going continue to refactor this cookbook until it supported multiple platforms. We could have started with that goal. Instead we started small. One test. One recipe. Refactor. Add more tests. Refactor. This process allowed us to deliver a reliable cookbook in confident way. But testing was not the only thing that aided us in building this cookbook.

Instrumental to software development and test-driven development is learning how to divide the work into these small increments. Small, deliverable, verifiable steps are essential to developing code with confidence. Now that you have seen and experienced the Test Driven Development (TDD) workflow and understand the basics, the real work that lay before you is to understand how to find these divisions in the requirements you are given.

This was a hand-picked experience. That moves we made may have seemed contrived. As with any knowledge transfer the best we can do is give you a model to play with and hope the forms hold true when it comes time for you to solve a problem with real requirements.

Slide 2

Objectives

After completing this module, you should be able to:

➢ Define expectations for multiple platforms
➢ Implement a cookbook that supports multiple platforms

In this module you will learn how to define expectations for multiple platforms and implement a cookbook that supports multiple platforms.

Slide 3

In this module we are going to develop solution in the opposite of the way we started. Instead of approaching this problem from the outside-in we are going to build it inside-out.

To do that means we are going to leverage the specifications we have written that validate the resources within our recipe. But before we do we need to gather some information that is important. Like the name of the platform we are using?

We could attempt to solve this problem by looking for documentation or a general search on the Internet. Instead we will ask the one source that knows the best: the executing code itself.

Slide 4

Add a Break Point to the Default Recipe

`~/httpd/recipes/default.rb`

```
#
# Cookbook Name:: httpd
# Recipe:: default
#
# Copyright (c) 2016 The Authors, All Rights Reserved.
require 'pry'                                                          +
binding.pry
include_recipe 'httpd::install'
include_recipe 'httpd::configuration'
include_recipe 'httpd::service'
```

To understand the platform of the node we simply need to set a break point in one of the recipes or the attributes file.

Slide 5

Execute the Tests to Initiate Pry

```
> chef exec rspec
```

```
From: /tmp/d20161027-28748-19ibu2o/cookbooks/httpd/recipes/default.rb @ line 7
Chef::Mixin::FromFile#from_file:

    2: # Cookbook Name:: httpd
    3: # Recipe:: default
    4: #
    5: # Copyright (c) 2016 The Authors, All Rights Reserved.
    6: require 'pry'
 => 7: binding.pry
    8: include_recipe 'httpd::install'
    9: include_recipe 'httpd::service'
```

8-5

Execute the tests.

Slide 6

Query the Node Object's Platform

```
[1] pry(#<Chef::Recipe>)> node.platform
```

```
"chefspec"
```

And then query the platform of the node object. The results should tell you that the platform for the node object in the ChefSpec environment is 'chefspec'.

Slide 7

REFERENCE

Fauxhai

ChefSpec by default uses a 'chefspec' platform. You can specify a platform from a list of platforms that are stored in a gem named 'Fauxhai'.

The gem contains static node objects for most major platforms and versions.

https://github.com/customink/fauxhai/tree/master/lib/fauxhai/platforms

https://github.com/customink/fauxhai

©2016 Chef Software Inc.   8-7   CHEF

The 'chefspec' platform is set by the ChefSpec gem. The platform has gone unspecified and this is what ChefSpec defaults to use. Now that we care about the platform we need to learn about another gem named Fauxhai. ChefSpec employs Fauxhai to provide fake node object data for various platforms.
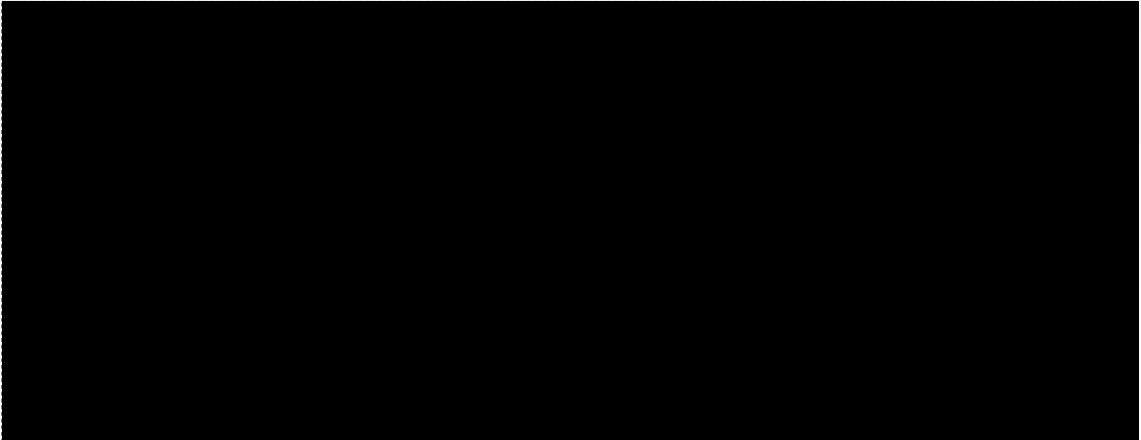
These platforms and their various versions are defined in the gem itself. Essentially the gem, at the time of writing this, contains a large number of JSON files which hold the node object results on each specific platform and version it supports. The best way to learn what platforms are provided is to read the source code in the Fauxhai repository.

Slide 8

Immediately Exit the Execution

```
[2] pry(#<Chef::Recipe>)> exit!
```

Now that we know the platform it is time to exit the execution.

Slide 9

Using Pry we were able to learn something about the system without having to rely on documentation. To understand the available platforms you have to rely on reading the source code.

Learning this powerful skill of gathering details will help you solve mysteries and provide more details and queries when searching for help on the Internet. The better you can get at understanding when to employ Pry and how to use it will eventually have you using documentation less and using executing code and source code more.

Instructor Note: Finding out which platforms and versions ChefSpec supported alluded me when first working with the project. There is some mention in the ChefSpec README but I believe I found myself diving into source code and stumbling upon the Fauxhai code. This is something that would be great to show to show learners if you are capable of figuring that out.

Slide 10

Remove the Break Point from the Recipe

`~/httpd/recipes/default.rb`

```
#
# Cookbook Name:: httpd
# Recipe:: default
#
# Copyright (c) 2016 The Authors, All Rights Reserved.
require 'pry'                                                        `
binding.pry
include_recipe 'httpd::install'
include_recipe 'httpd::service'
```

©2016 Chef Software Inc.   8-10   CHEF

It is a good habit to clean up this break points. Leaving them around has a nasty habit of pausing the execution of a run you want to see complete uninterrupted.

Slide 11

Now that we know the environment it is time to get to work on defining those new examples for the new platform that we want to support.

Slide 12

EXERCISE

Support for CentOS & Ubuntu

The best of both worlds!

Objective:

❑ Write a test that verifies the Install recipe chooses the correct package on CentOS & Ubuntu

❑ Execute the tests and verify the tests fail

❑ Update the attribute to provide support for CentOS & Ubuntu

❑ Execute the tests and verify the tests pass

©2016 Chef Software Inc.   8-12

Together let's walk through refactoring the cookbook's install recipe. Like we have done before. When we are done it will be your turn to implement the solution for the remaining recipes.

Slide 13

Update the Context to be Platform Specific

`~/spec/unit/recipes/install_spec.rb`

```
describe 'httpd::install' do
  context 'When all attributes are default, on CentOS' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new(platform: 'centos', version: '6.7')
      runner.converge(described_recipe)
    end

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end

    it 'installs the appropriate package' do
      expect(chef_run).to install_package('httpd')
    end
  end
  # ... SPECIFICATION CONTINUES ON THE NEXT SLIDE ...
```

First we will start by updating our current specification. The context up to this point has been 'on an unspecified platform'. We want to instead state that these first two examples are for the CentOS platform. That change is purely cosmetic.

The change that matters is the one in which we provide new parameters to the ServerRunner class initializer that state the specific platform and version we are interested in verifying against. If we specify an unsupported platform or platform version we will see an error when the tests execute. This is again why it is important to review the Fauxhai project.

Slide 14

Execute the Tests to See it Pass

```
> chef exec rspec spec/unit/recipes/install_spec.rb

..

Finished in 1.35 seconds (files took 4.51 seconds to load)
2 examples, 0 failures
```

©2016 Chef Software Inc.   8-14

Because we made changes the original expectations it might be a good moment to execute the tests and ensure that everything is still working for the CentOS platform.

Slide 15

Add a Second Context for Another Platform

`~/spec/unit/recipes/install_spec.rb`

```
# ... CONTINUED FROM THE PREVIOUS SLIDE ...
context 'When all attributes are default, on Ubuntu' do
  let(:chef_run) do
    runner = ChefSpec::ServerRunner.new(platform: 'ubuntu', version: '14.04')
    runner.converge(described_recipe)
  end

  it 'converges successfully' do
    expect { chef_run }.to_not raise_error
  end

  it 'installs the necessary package' do
    expect(chef_run).to install_package('apache2')
  end
end
end
```

©2016 Chef Software Inc.   8-15

Now return to the specification file and alongside CentOS example group it is time to define the example group that will contain the examples for the Ubuntu 14.04 platform.

The format is nearly identical between these two example groups save for the context, the parameters specified to the ServerRunner initialization, and the name of the necessary package to install.

Slide 16

The examples have now been defined for the existing platform and the new platform.

Slide 17

Execute the Tests to See it Fail

```
> chef exec rspec spec/unit/recipes/install_spec.rb
```

```
...F

Failures:

  1) httpd::install When all attributes are default, on Ubuntu
installs the appropriate package
      Failure/Error: expect(chef_run).to install_package('apache2')
        expected "package[apache2]" with action :install to be in
Chef run. Other package resources:

        apt_package[httpd]
```

©2016 Chef Software Inc.   8-17

When it comes time to execute the tests we should see that defining the new platform will not raise an error when it converges but will fail to meet the expectation that we installed the correctly named package.

Slide 18

EXERCISE

Support for CentOS & Ubuntu

Failure means we have work to do!

Objective:

- ✓ Write a test that verifies the Install recipe chooses the correct package on CentOS & Ubuntu
- ✓ Execute the tests and verify the tests fail
- ❑ Update the attribute to provide support for CentOS & Ubuntu
- ❑ Execute the tests and verify the tests pass

©2016 Chef Software Inc.   8-18

The name of the package is defined in the attributes file. That is what we refactored to support in the last section. It is now time to return to the attributes file and have it specify a different package name based on the platform.

Slide 19

CONCEPT

Switching on Node Platform

To control the flow of execution we need to employ some Ruby conditional statements. Conditional statements allow us to alter this control flow. Because we have access to the power of Ruby we have many choices.

https://docs.chef.io/dsl_recipe.html#case-statements

©2016 Chef Software Inc.   8-19

To set the node attribute conditionally based on the platform means we are going to need to control the way that Ruby parses the code based on the state of the node platform. Ruby provides many ways to control the flow and several of them are documented in the recipe Domain Specific Language (DSL).

Slide 20

Update the Attributes to Support Platforms

`~/httpd/attributes/default.rb`

```ruby
case node['platform']
when 'ubuntu'
  default['httpd']['package_name'] = 'apache2'
else
  default['httpd']['package_name'] = 'httpd'
end

default['httpd']['package_name'] = 'httpd'
default['httpd']['service_name'] = 'httpd'
default['httpd']['default_index_html'] ='/var/www/html/index.html'
```

©2016 Chef Software Inc.   8-20

A very common way is to define a case statement. The case statement allows you to provide a value or value stored in a variable to the case keyword. Then following the case statement are a number of 'when' statements. Each 'when' needs to be provided with a value or value stored in a variable. If the value in the case statement equals the value in when statement then it is match and the flow of execution will take that path and ignore all others.

If none were to match we might be in trouble as the node attribute would never be set so we can use an 'else' statement which is as good as saying if none of those match then use this path.

The order of the case statement is particularly important as well. The first match that is made is the path the execution will take.

Instructor Note: When we say 'equal' each other we mean that Ruby is comparing the objects together with the equality method, the triple equals (===) .

Slide 21

EXERCISE

Support for CentOS & Ubuntu

This should do it!

Objective:

- ✓ Write a test that verifies the Install recipe chooses the correct package on CentOS & Ubuntu
- ✓ Execute the tests and verify the tests fail
- ✓ Update the attribute to provide support for CentOS & Ubuntu
- ❑ Execute the tests and verify the tests pass

©2016 Chef Software Inc.   8-21

Now that the attributes file has been updated it is time execute the tests again and see if we defined this conditional logic correctly.

Slide 22

Execute the Tests to See it Pass

```
> chef exec rspec spec/unit/recipes/install_spec.rb
```

```
....

Finished in 1.35 seconds (files took 4.51 seconds to load)
4 examples, 0 failures
```

Executing the tests we should see both platforms will converge without error and install the necessary packages.

Slide 23

EXERCISE

Support for CentOS & Ubuntu

Woot! Multi-platform support for the installation!

Objective:

✓ Write a test that verifies the Install recipe chooses the correct package on CentOS & Ubuntu

✓ Execute the tests and verify the tests fail

✓ Update the attribute to provide support for CentOS & Ubuntu

✓ Execute the tests and verify the tests pass

©2016 Chef Software Inc.   8-23

This approach to leverage the existing examples and use them to help define new examples for a new platform allowed us to build confidence through testing from the inside-out.

Taking this inside-out approach can feel right in situations where you know the steps you have to take.

Slide 24

Now as an exercise for you it is time to do the same thing for the service recipe. The service for Ubuntu is named 'apache2'. Start with the changes to the specifications, move through see the failure, update to use the same conditional statement structure and then see the examples verify your work.

Instructor Note: Allow 10 minutes to complete this exercise

Slide 25

Update the Context to be Platform Specific

`~/spec/unit/recipes/service_spec.rb`

```ruby
describe 'httpd::service' do
  context 'When all attributes are default, on CentOS' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new(platform: 'centos', version: '6.7')
      runner.converge(described_recipe)
    end
    # ... it converges successfully ...

    it 'starts the appropriate service' do
      expect(chef_run).to start_service('httpd')
    end

    it 'enables the appropriate service' do
      expect(chef_run).to enable_service('httpd')
    end
    # ... SPECIFICATION CONTINUES ON THE NEXT SLIDE ...
```

©2016 Chef Software Inc.   8-25

Let's review the changes to the service specification. You start with ensuring the existing CentOS platform is explicitly stated in the context and defined in the parameters provided to the ServerRunner initialization.

Slide 26

Add a Second Context for Another Platform

`~/spec/unit/recipes/service_spec.rb`

```
  # ... CONTINUED FROM THE PREVIOUS SLIDE ...
  context 'When all attributes are default, on Ubuntu' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new(platform: 'ubuntu', version: '14.04')
      runner.converge(described_recipe)
    end
    # ... it converges successfully ...

    it 'starts the appropriate service' do
      expect(chef_run).to start_service('apache2')
    end
    it 'enables the appropriate service' do
      expect(chef_run).to enable_service('apache2')
    end
  end
end
```

You now define an entire example group dedicated to the Ubuntu platform which defines the same structure of examples but with the values that are important for the platform.

Slide 27

Execute the Tests to See it Fail

```
> chef exec rspec spec/unit/recipes/service_spec.rb
```

```
....FF


Failures:

  1) httpd::service When all attributes are default, on Ubuntu
starts the appropriate service
     Failure/Error: expect(chef_run).to start_service('apache2')
       expected "service[apache2]" with action :start to be in
Chef run. Other service resources:

       service[httpd]
```

©2016 Chef Software Inc.   8-27

Executing the test you would see the appropriate failures for the correctly named services not being started and enabled.

Slide 28

Update the Attribute to Support Platforms

`~/httpd/attributes/default.rb`

```
case node['platform']
when 'ubuntu'
  default['httpd']['package_name'] = 'apache2'
  default['httpd']['service_name'] = 'apache2'
else
  default['httpd']['package_name'] = 'httpd'
  default['httpd']['service_name'] = 'httpd'
end

default['httpd']['service_name'] = 'httpd'
default['httpd']['default_index_html'] = '/var/www/html/index.html'
```

©2016 Chef Software Inc.   8-28

Updating the attributes for the service should be a little less work because the structure is all in place.

Slide 29

Execute the Tests to See it Pass

```
> chef exec rspec spec/unit/recipes/service_spec.rb
```

```
......

Finished in 1.84 seconds (files took 4.22 seconds to load)
6 examples, 0 failures
```

©2016 Chef Software Inc.   8-29

Finally when we execute the tests again we see that all the examples pass.

Slide 30

Support for CentOS & Ubuntu

- ✓ Write a test that verifies the Service recipe chooses the service named 'httpd' on CentOS and 'apache2' on Ubuntu
- ✓ Execute the tests and verify the tests **fail**
- ✓ Update the attribute to choose the service name 'httpd' on CentOS and 'apache2' on Ubuntu
- ✓ Execute the tests and verify the tests **pass**

©2016 Chef Software Inc.   8-30

That was nearly identical and a good way to reinforce the testing flow.

Slide 31

Now only the configuration recipe remains. The default index HTML page for Ubuntu and CentOS are exactly the same. So when you define the new examples you actually will not see the failure. Then when you make the changes to the attributes file you will not see the failure. At that point you have written two new examples for the Ubuntu platform and it is important to ensure those tests fail. So pick a mutation (e.g. remove a line or specify an incorrect value) for the Ubuntu flow and ensure you see the failure.

Finally take a look at the code that you have created and ask yourself is that change better?

Instructor Note: Allow 10 minutes to complete this exercise

Slide 32

Update the Context to be Platform Specific

`~/spec/unit/recipes/configuration_spec.rb`

```ruby
describe 'httpd::configuration' do
  context 'When all attributes are default, on CentOS' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new(platform: 'centos', version: '6.7')
      runner.converge(described_recipe)
    end
    # ... it converges successfully ...

    it 'creates a default index html page' do
      expect(chef_run).to create_file('/var/www/html/index.html')
    end

# .... SPECIFICATION CONTINUES ON THE NEXT SLIDE ...
```

Same as before we start with some maintenance of CentOS examples.

Slide 33

Add a Second Context for Another Platform

`~/spec/unit/recipes/configuration_spec.rb`

```ruby
# ... CONTINUED FROM THE PREVIOUS SLIDE ...
context 'When all attributes are default, on Ubuntu' do
  let(:chef_run) do
    runner = ChefSpec::ServerRunner.new(platform: 'ubuntu', version: '14.04')
    runner.converge(described_recipe)
  end
  # ... it converges successfully ...

  it 'creates a default index html page' do
    expect(chef_run).to create_file('/var/www/html/index.html')
  end
end
end
```

You then define another example group dedicated to the Ubuntu platform. Except this time the expectation is exactly the same.

Slide 34

Execute the Tests to See it Pass

```
> chef exec rspec spec/unit/recipes/configuration_spec.rb
```

```
....

Finished in 1.84 seconds (files took 4.22 seconds to load)
4 examples, 0 failures
```

Executing the tests shows you that everything is working.

Slide 35

Update the Attribute to Support Platforms

`~/httpd/attributes/default.rb`

```
case node['platform']
when 'ubuntu'
  default['httpd']['package_name'] = 'apache2'
  default['httpd']['service_name'] = 'apache2'
  default['httpd']['default_index_html'] = '/var/www/html/index.html'
else
  default['httpd']['package_name'] = 'httpd'
  default['httpd']['service_name'] = 'httpd'
  default['httpd']['default_index_html'] = '/var/www/html/index.html'
end

default['httpd']['default_index_html'] = '/var/www/html/index.html'
```

You implemented the change that we have done before.

Slide 36

Execute the Tests to See it Pass

```
> chef exec rspec spec/unit/recipes/configuration_spec.rb
```

```
....

Finished in 1.84 seconds (files took 4.22 seconds to load)
4 examples, 0 failures
```

And finally see the tests pass again. This is where you should become uncomfortable that we may have a false positive and that is a good time to ensure that you do not by mutating the code.

Slide 37

Update the Attribute to Support Platforms

`~/httpd/attributes/default.rb`

```
case node['platform']
when 'ubuntu'
  default['httpd']['package_name'] = 'apache2'
  default['httpd']['service_name'] = 'apache2'
  default['httpd']['default_index_html'] = '/var/www/html/index.html2'
else
  default['httpd']['package_name'] = 'httpd'
  default['httpd']['service_name'] = 'httpd'
  default['httpd']['default_index_html'] = '/var/www/html/index.html'
end
```

8-37

So anywhere in the Ubuntu flow of execution make a small mutation. In the example I am providing I have chosen a different path. Removing the attribute is another option as well.

Slide 38

Execute the Tests to See it Pass

```
> chef exec rspec spec/unit/recipes/configuration_spec.rb

...F

Finished in 1.84 seconds (files took 4.22 seconds to load)
4 examples, 1 failures
```

Executing the tests should net at least one failure and that should give you more confidence that the expectations you have written are doing the work you want them to do.

Slide 39

Update the Attribute to Support Platforms

`~/httpd/attributes/default.rb`

```
case node['platform']
when 'ubuntu'
  default['httpd']['package_name'] = 'apache2'
  default['httpd']['service_name'] = 'apache2'
  default['httpd']['default_index_html'] = '/var/www/html/index.html'
else
  default['httpd']['package_name'] = 'httpd'
  default['httpd']['service_name'] = 'httpd'
  default['httpd']['default_index_html'] = '/var/www/html/index.html'
end

default['httpd']['default_index_html'] = '/var/www/html/index.html'
```

©2016 Chef Software Inc.   8-39

Finally you might restore the code. Removing the mutation.

You may even choose to undo the change the proposed change. This is up to you to make the decision. In the example shown here I have returned to the original implementation. The original implementation worked, executing our tests proved it. Whether you should leave the attribute defined in the case statement or outside of it is up to you.

Leaving it in the case statements ensures that all values are defined on the platform. If a value on a particular platform were to change we would simply need to only change it within that platform's flow of control. However, if you never implement another platform you have created two lines of code. Some may argue the fewer lines of code you issue or statements you place inside of a conditional make it easier to read and understand.

The most important thing is that the examples you defined should remain in the specification regardless of the implementation. The examples describe the expected behavior of the platform.

Slide 40

Support for CentOS & Ubuntu

✓ Write a test that verifies the file recipe chooses the same path (name) '/var/www/html/index.html' on CentOS and on Ubuntu

✓ Execute the tests that verify the tests **pass**

✓ Update the attribute to choose the same path on CentOS and on Ubuntu

✓ Execute the tests that verify the tests **pass**

✓ Get nervous! Mutate the attributes file!

✓ Undo the entire attributes change and verify the tests **pass**

There is only one more thing to do.

©2016 Chef Software Inc.   8-40

Congratulations!

Slide 41

PROBLEM

What About an Integration Test

Remember that ChefSpec and Fauxhai are fake in-memory representations of a chef-client run. They are not equivalent to running the recipe on the specified platform.

©2016 Chef Software Inc.   8-41

CHEF

Now that we have finished building everything from the inside-out. It is finally time to see if the integration test works. This is important. When building recipes with ChefSpec you can very quickly make mistakes. Those mistakes are not the typos or omissions we have made. These are the mistakes that only the platform can catch.

Because we have been doing everything in-memory we really do not know if the package name, file path, or service name actually works. The only way to prove that is to apply the recipe to that platform.

Slide 42

EXERCISE

Integration Test with Ubuntu

This is where it all started.

Objective:

❑ Update the Kitchen Configuration to test on Ubuntu
❑ Execute the integration tests and verify that they pass

8-42

So for our last and final exercise together lets update the Kitchen configuration to give us the ability to test on the Ubuntu platform.

Slide 43

Add a New Platform to the Kitchen Configuration

~/httpd/.kitchen.yml

```
---
driver:
  name: docker

provisioner:
  name: chef_zero

platforms:
  - name: centos-6.7
  - name: ubuntu-14.04

suites:
  - name: default
    run_list:
      - recipe[httpd::default]
    attributes:
```

8-43

Within the kitchen configuration we define the new Ubuntu 14.04 platform.

Slide 44

Verify the New Instance is Present

```
> kitchen list
```

Instance	Driver	Provisioner	Verifier	Transport	Last Action
default-centos-67	Docker	ChefZero	Busser	Ssh	Set Up
default-ubuntu-1404	Docker	ChefZero	Busser	Ssh	<Not Created>

©2016 Chef Software Inc.   8-44

We verify that the platform exists within the list of instances.

Slide 45

EXERCISE

Integration Test with Ubuntu

Fingers crossed

Objective:

✓ Update the Kitchen Configuration to test on Ubuntu
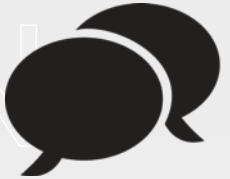❑ Execute the integration tests and verify that they pass

©2016 Chef Software Inc.   8-45

And now it is time to execute the test suite. By choosing a very valuable and implementation free ServerSpec example, is the website up and running in localhost, we can be fairly certain that the expectations should be met.

Slide 46

Execute the Tests for All Platforms

```
> kitchen test

-----> Starting Kitchen (v1.4.2)
-----> Cleaning up any prior instances of <default-centos-67>
-----> Destroying <default-centos-67>...
       Finished destroying <default-centos-67> (0m0.00s).
-----> Testing <default-centos-67>
-----> Creating <default-centos-67>...
```

©2016 Chef Software Inc.   8-46

To execute the tests against both platforms run 'kitchen test'. Because we have two instances and did not specify a particular instance with the command it will run tests against all the listed instances.

This might be a good time to get up and move around as it will take some time.

Slide 47

The expectations should pass and this brings the last exercise to a close.

Let's have a discussion.

Slide 48

8-48

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.

Slide 49

Before we complete this section, let us pause for questions.

Slide 50

Thank you for your time and attention.

Slide 51